Free and Open Source Software

Dirk Riehle, Friedrich-Alexander University Erlangen-Nürnberg

Free software is software that gives users the right to use the software, to modify the software, and to pass on the software, modified or not, all free of charge and without restrictions on what the software is used for. Open source software provides users with the same rights as free software. For all practical purposes, they are the same.

The term *free software* was used already in the late 1970s, but was codified by Richard Stallman through the definition of the four software freedoms in 1986. The definition and publication was performed through the Free Software Foundation, a U.S.-based nonprofit organization founded by Stallman in 1985.

The term *open source software* was defined by the Open Source Initiative, a U.S.-based nonprofit organization founded in 1998 by Bruce Perens and Eric Raymond. The definition was written by Perens and consists of a list of 10 criteria, which Perens derived from the Debian Free Software guidelines.

Open source software is often abbreviated as OSS and free and open source software is often abbreviated as FOSS. Sometimes the term *libre* is thrown in to emphasize the freedom to do what you want and to downplay that the software is free of charge. This leads to free/libre, and open source software, abbreviated as FLOSS. This term is primarily used by nonnative speakers of English.

Not everyone agrees that free and open source software are the same. Free software proponents argue that users who receive free software must also be given access to its source code, even if modified by the provider of the software. To enforce the right of a user to receive the source code, Stallman invented the copyleft obligation, which requires that anyone who distributes free software cannot change the license terms. This prevents software vendors from keeping proprietary modifications of free software locked up.

Rights to and obligations for free and open source software are codified as free and open source software licenses. The copyleft obligation became popular as part of the GNU General Public License 2.0, a prominent free and open source software license.

Digital Object Identifier 10.1109/MC.2024.3407268
Date of current version: 26 July 2024

EDITOR DIRK RIEHLE
Friedrich Alexander-University of Erlangen Nürnberg;
dirk.riehle@fau.de

In contrast to free software, open source proponents tend to rely on enlightened self-interest of software users to contribute their modifications to open source software projects. They don't try to force anyone who distributes open source software to lay open any modifications if they don't want to.

Open source licenses that grant the rights listed above but don't contain a copyleft obligation, are called *permissive licenses*, while licenses that contain a copyleft obligation are called *copyleft licenses*. Common examples of permissive licenses are the Massachusetts Institute of Technology (MIT) license, the Apache 2.0 license, and the BSD family of licenses.

For the better part of the 1990s and the 2000s, the philosophical debate about software freedom and copyleft allowed the enemies of free and open source software to spread fear, uncertainty, and doubt about the usefulness of open source software and arguably delayed its dominance by a decade or two.

Today, free and open source software are part of almost all existing software, closed or not.

FROM THE EDITOR

This month's article on open source software covers the basics: What free and open source software is, why people and companies are using it, how open source licenses work, and of course, what the copyleft obligation is. This article starts a thread of explanatory articles about open source software in this column. This thread will be written by me, and it will interrupted by research and industry contribution articles as I'm able to acquire them. (Keep them coming!) As part of engaging with you, dear reader, you can comment on this article at https://uni1.de/ose01. I'm looking forward to your comments. Until then, keep on hacking!—*Dirk Riehle*

WHY USE OPEN SOURCE SOFTWARE?

Open source software succeeded because of the benefits it provides to users and despite the challenges it poses.

The main benefit of using open source software for a user is that they avoid vendor lock-in. They can use the software under defined and beneficial circumstances (the open source license) and do not depend on a vendor.

The lack of vendor lock-in creates the following three more specific reasons for using open source software in projects and products:

- *Free of charge:* As already mentioned, open source software can be used free of charge. There are no license fees. While the use of open source software can create secondary costs (for example, maintenance costs), the total cost of using the software is usually much lower than licensing a closed source software from a vendor.
- *Option to adapt to your needs:* Open source software is available in source code form and comes with the right to adapt the software to your needs. If you were locked-in to some vendor's closed-source software, you'd have to ask and pay them for any modifications you might need, and there is no guarantee that they will create those modifications for you. With open source software, you can simply make the necessary changes yourself or hire someone to do it for you.
- *Operational safety:* Open source software does not come with an end date. The usage rights are given to you forever. Even if you buy a perpetual license from a vendor and start using a closed source software in projects and products, the vendor might still discontinue the software or even go out of business. There is no guarantee that the software will be maintained, no guarantee that bugs will be fixed, etc. Open source, in contrast, will always be available as long as there are copies, and you can always help yourself. This benefit is particularly important for long-lived products, like mobility solutions (cars, trains, planes, etc.), because these products easily outlive the suppliers of the software components to these long-lived products.

There are many other reasons why people use and also contribute to open source software. They may be learning something of interest to them, and they are contributing to software as a shared common good. Some even argue that open source project communities will ultimately be more innovative than closed source software vendors.

There are many more economic reasons, beyond avoiding vendor lock-in, why other vendors should be using, contributing to, and leading open source projects. They are discussed in more detail in a later column. Before we can get there, we have to discuss, however, how open source works.

INTELLECTUAL PROPERTY RIGHTS

Property is, by definition, something that has an owner. The owner of some property determines what can be done with the property, even destroy it. A person's property is protected by laws to ensure that society and the economy work well and without disruption. Examples of physical property are the Eiffel tower owned by the city of Paris, or a cruise ship owned by a cruise company, or the mobile phone owned by you.

Intellectual property is an intangible form of property: It exists only in our minds, it is virtual or digital, an idea or a creative expression. Examples of intellectual property are the William Gibson novel *Idoru*, the Microsoft brand logo, or the CRISPR/Cas patents on gene editing. Like physical property, intellectual property is protected by laws. These laws are expressed as the intellectual property rights that an owner is given.

Software is a form of intellectual property. The owner can determine what others can do with it, for example, to use the software. Implied by ownership is that nobody else is allowed to use some property without the owner's agreement. For this reason, property rights are also often called *exclusion rights*. Users of the software typically pay a so-called *license fee* to the owner for them to waive their exclusion rights to their property.

The three most important intellectual property rights are:

- *Copyright:* Copyright is the ownership right to a specific written expression (like a novel); it is not about the idea behind the expression. Therefore, a program written in Java can be rewritten in Python and it will have a different copyright. Each program potentially has a different owner. Copyright is granted automatically, upon creation. There is no need to register a copyright. Copyright expires, but only after many decades (depending on the jurisdiction).
- *Patent rights:* A patent is a nontrivial man-made invention. A patent right is an ownership right to an invention to exploit it as one sees fit. The patent is about the idea, for example, a mechanism. The same mechanism, realized in two different ways, still falls under the same patent. Patent rights need to be registered by application for the patent at a patent office. If granted, a patent right holds for several decades (shorter than copyright though) and eventually expires as well.
- *Trademark rights:* A trademark is a specific mark representing some other property, and a trademark right is the ownership right of that trademark. Examples of marks are visual signs (logos) or sound marks (like the Nokia jingle), and even smell marks: that is, uniquely identifiable smells. Trademark rights are granted automatically through the creation of the mark and they are maintained by enforcing the mark's ownership. Trademarks can be registered. They live as long as the owner enforces their ownership to the mark; once they stop fighting uses by others, they lose the trademark rights.

There are other forms of intellectual property, like trade secrets (for example, customer lists), but they are not needed here. The specifics vary greatly, typically by jurisdiction. Some countries don't have copyright laws or have different rules for patent and trademark rights. Due to the global nature of the software business, the intellectual property rights granted by leading countries or unions, like the United States or the European Union, affect everyone, though, and need to be understood and managed.

Closed source software, like Adobe Photoshop, is typically affected by all intellectual property rights. There will be the primary copyright owner (Adobe), but also many other copyright owners. Other copyright owners are the developers of the non-Adobe components that Photoshop has been built from, including open source software. In all likelihood, there are many patents implemented in Photoshop, owned by Adobe and third parties. Finally, the Photoshop logo is a trademark owned by Adobe.

There are few nontrivial software applications today that do not have a large number of owners of the different intellectual properties embedded in the software. Each developer of such an application needs to understand how and why they are incorporating other parties' intellectual property in their software. This is often a laborious task.

OPEN SOURCE SOFTWARE LICENSES

Licenses are a contract (in most jurisdictions) between a licensor and a licensee. Typically, the licensor allows the licensee to use their property in return for some payment.

Software licenses grant licensees some rights to the software. Software licenses can be limited in many ways: There may only be one user allowed at-a-time, the software may only be used until the end of the year, the software may not be used outside of Germany, etc. If someone says they bought some software, what they typically did was to assume the licensee role and pay the licensor a fee for a usage right to the software.

Open source software licenses are licenses that have been approved by the Open Source Initiative. The Open Source Initiative is a U.S.-based nonprofit organization that serves as a spokesperson for and an arbiter of what open source means.

Open source (software) licenses all follow a common pattern. They consist of four main sections:

- *The rights grant:* An open source license always (by definition) grants the licensee the right to use the software, to receive the source code, to modify the source code and run the modified program, and to distribute the source code and resulting programs in unmodified or modified form.
- *The obligations:* An open source license may impose obligations on the licensee. For example, if someone passes on the open source software, they may have to create and provide legal notices of the open source software to recipients. Obligations vary widely between licenses.

- *The prohibitions:* An open source license may contain clauses that tell a licensee what they are not allowed to do. Typically, a licensee may not claim endorsement of any uses of the open source software by the licensors of the open source code or the creators of the license texts.
- *The disclaimer:* Most open source licenses disclaim warranties and liabilities. Licensors try to make sure that anything bad that happens through the open source code does not fall back on them. Such disclaimers may be limited by the laws of the jurisdiction where they apply.

Software becomes open source software if the owner of the software decides to license out the software to the world using an open source license. Thus, the status of open source is not an intrinsic property of the software, but is decided by the original owner, who decides to become an open source software licensor.

Across all open source licenses, the rights grant is always the same, whatever the specific choice of words. Prohibitions, if any, and the disclaimers are also almost always the same. Obligations, however, differ widely between licenses.

The most important common obligations are:

- *Provision of legal notices:* An open source user, upon distribution of a binary version of the open source code, must compile all relevant legal notices (license texts, copyright statements, and other notices) and provide them to the recipient of the code.
- *The copyleft obligation:* An open source user, when distributing the open source code, must apply the license of the incoming open source code to the outgoing code, including any proprietary modifications made to it, or lose the right to use and distribute the code.
- *Indemnification:* An open source user, who distributes open source code, has to help defend and indemnify any open source developer whose code they are distributing, if a recipient of the user's distribution chooses to take legal action against the developer.

Legal notices and the copyleft obligation will be discussed in later sections of this column.

The Linux Foundation is an industry-led nonprofit organization furthering open source projects. Its SPDX project is giving us unique identifying names for established open source licenses. These so-called *SPDX identifiers* encode, in human-readable form, the name and version of a license and sometimes specific conditions.

- An example of a simple SPDX identifier is MIT for the MIT license.
- An example of a versioned SPDX license identifier is EPL-2.0 for the Eclipse 2.0 license.
- An example of a complex SPDX license identifier is AGPL-3.0-or-later for the Affero GPL 3.0 license with the option to choose a successor license.

Open source license texts themselves are legal documents. Typically, they are free to use. Some allow modification—for example, the Apache-2.0 license—some don't, for example, the GPL-2.0 license.

END-USERS AND DISTRIBUTORS

In open source licenses, there are two types of users: end-users and distributors.

- *End-users:* An end-user of an open source software receives the software but does not pass it on. They are the final element in a chain of receiving and passing on the software.
- *Distributors:* A distributor receives the open source software and also passes it on to third parties. They are the intermediate elements in the chain of receiving and passing on the software.

This distinction has an important consequence. There are typically no license obligations for an end-user. All the obligations are put upon the distributor, who provides the software to a third party, for example, a customer. Only a distributor has to worry about provision of legal notices, the copyleft obligation, or indemnification.

The terms *end-user* and *distributor* don't show up in the license texts; however, they are commonly used to describe the two situations laid out in the licenses. Almost all licenses distinguish between a situation where the recipient of an open source software uses it for themselves (end-user), and a situation where the recipient of the open source software passes the software on to third parties (distributor).

Companies often explain this distinction as open source use cases to their employees so that it becomes easier to identify whether the open source software will be distributed and whether the corresponding license obligations kick in.

A person or company is an end-user if the software is not passed on further. Examples are:

- *In-house use of the open source software*, for example, editing documents using LibreOffice or compiling source code using gcc.
- *Demonstration of sales prototypes* that include open source components, as long as the demo code isn't given to third parties.
- *Operation of open source software as a cloud service*, as long as the open source code doesn't leave the (public or private) cloud.

If a person or company distributes code to third-parties, they are a distributor. Examples are:

- *Provision of a software to be deployed on-premise*, including

traditional laptop or workstation applications, mobile apps, etc. to customer or other third parties.
- *Provision of website code that gets downloaded into a user's browser* (mostly Javascript, HTML, CSS), if that user is a customer or other third party.
- *Provision of container images*, including in recipe form (for example, Dockerfiles) and through registries, to customers or other third parties.

Open source software can be an application, for stand-alone use, or a component, for incorporation into a product. People or companies who use an application for their own purposes are obviously end-users. Companies who include open source components in a product they sell are obviously distributors.

People and companies can both be end-users or distributors.

THE COPYLEFT OBLIGATION

A particularly important obligation is the copyleft obligation found in some, but not all, open source licenses. The copyleft obligation requires that any incoming open source code with a copyleft obligation be distributed under the same license to third parties only. In short: The outgoing license must be the same as the incoming license.

This does not only apply to the original open source code, but also to any code that is derived from the incoming copyleft-licensed open source software. A distributor cannot take incoming copyleft-licensed code, modify it, and distribute it under their own proprietary license. It must be distributed under the incoming copyleft license.

Not all closed source software will be affected by incoming copyleft-licensed code. U.S. copyright law makes a distinction between derivative and collective works. The copyleft effect applies only to derivative, but not to collective works. Applied to software, it leads to the following definitions:

- *Derivative code* is code created by building on the original code in such a way that the original code cannot be separated from any additions and modifications easily and using standard tools. The prime example of derivative code is code created by modifying the original code.
- *Collective code* is a set of codes (programs, libraries) that are kept separate from each other and where each individual code is accessible using standard tools. The prime example of a collective work is when distributors put independent programs next to each other into the same directory.

Developers usually prefer not to modify other people's code. Rather, they'll try to use it as a library. However, using a copyleft-licensed library makes any using code a derivative of the library, because the using code necessarily incorporates the interface symbols of the library. This seemingly little beachhead is enough to turn the using code into derivative code, even if the using code and used library are maintained as separate files.

Licenses that do not contain a copyleft obligation are called *permissive licenses*. Licenses that contain a copyleft obligation can be split into *weak* and *strong* copyleft licenses. Weak copyleft licenses weaken the copyleft effect by stopping it at a code boundary, if the copyleft-licensed code is its own separate component that can be accessed using standard tools. An example is a dynamically linked library. In summary:

- *A permissive license* is a license that has no copyleft obligation. Examples are the MIT license, the Apache-2.0 license, and the BSD family of licenses.
- *A (strong) copyleft license* is a license with an unrestricted copyleft obligation. Examples are the GPL-2.0-or-later and the AGPL-3.0-or-later families.
- *A weak copyleft license* is a license where the copyleft effect stops at the code boundary, if the boundary "strongly separates" the using code from the used copyleft-licensed code. Two components are strongly separated if they can be accessed and modified independently of each other using standard tools. The main example of a weak copyleft license is the LGPL-2.0-or-later license.

Sometimes developers construct a weak copyleft license by taking a strong copyleft license and modifying or amending it to weaken it for a particular use case. The prime example is the Syscall Note by Linus Torvalds, through which he stopped the copyleft effect of the Linux kernel code from reaching application code that only uses regular application functions of the kernel.

The copyleft obligation was first introduced by the GPL-2.0-or-later license in 1991 and has proved divisive to the free and open source software world. Some applaud the obligation for its intent to ensure freedom for end-users, who will always have a right to receive the source code to any copyleft-licensed binary code they are receiving, while others have chided this intention and likened copyleft-licensed software to viruses. Given that most software vendors keep the source code for their products closed, they will try to keep copyleft-licensed code out of their code base at all costs. C

DIRK RIEHLE is a professor of computer science, specializing in software engineering and open source, with Friedrich-Alexander University Erlangen-Nürnberg, 91054 Erlangen, Germany. Contact him at dirk@riehle.org.